Published in final edited form as:
Conf Proc IEEE Eng Med Biol Soc. 2012 ; 2012: 2505–2508. doi:10.1109/EMBC.2012.6346473.

The Unlock Project: A Python-based framework for practical brain-computer interface communication “app” development

Jonathan S. Brumberg,
Department of Speech-Language-Hearing, University of Kansas, Lawrence, KS and the Department of Speech, Language and Hearing Sciences, Boston University, Boston, MA

Sean D. Lorenz,
Program in Cognitive and Neural Systems, Boston University, Boston, MA

Byron V. Galbraith, and
Program in Cognitive and Neural Systems, Boston University, Boston, MA

Frank H. Guenther
Graduate Program in Neuroscience and Department of Speech, Language and Hearing Sciences, Boston University, Boston, MA

Abstract

In this paper we present a framework for reducing the development time needed for creating applications for use in non-invasive brain-computer interfaces (BCI). Our framework is primarily focused on facilitating rapid software “app” development akin to current efforts in consumer portable computing (e.g. smart phones and tablets). This is accomplished by handling intermodule communication without direct user or developer implementation, instead relying on a core subsystem for communication of standard, internal data formats. We also provide a library of hardware interfaces for common mobile EEG platforms for immediate use in BCI applications. A use-case example is described in which a user with amyotrophic lateral sclerosis participated in an electroencephalography-based BCI protocol developed using the proposed framework. We show that our software environment is capable of running in real-time with updates occurring 50–60 times per second with limited computational overhead (5 ms system lag) while providing accurate data acquisition and signal analysis.

I. INTRODUCTION

Brain-computer interfaces (BCI) for communication and environmental interaction are entering a critical period in translation from research to practice. For years, researchers have shown how both invasive and non-invasive techniques for actuating external devices using brain activity is feasible, reliable and general enough for a range of potential users. However, the primary source of software development for BCI applications has been limited to a small minority directly involved in BCI research. We recognize this bottleneck as a critical impediment to BCI development and eventual widespread use. In an effort to alleviate this bottleneck, we propose a software development framework based upon a standardized application programming interface (API) designed to facilitate rapid “app” development. In particular, we focus on: 1) generalized data acquisition, 2) stimulus presentation (if applicable) and 3) user feedback (with most emphasis on # 3).

The need for a common standard in BCI development was first met by BCI2000, a well known BCI development environment [1] written and maintained by researchers at the Wadsworth Center (Albany, NY). The system employs a modular software development approach in which BCI developers can specify source (e.g. data acquisition), signal processing, user application and operator modules (e.g. intermodule communication).

BCI2000 is implemented in the C/C++ programming language and distributed as an open-source software package with a C/C++ application programming interface (API). Open-source developmental packages, such as this, benefit from the fact that many users have access to software source code and documentation. Such widespread availability facilitates cooperation and collaboration between distant research groups ultimately leading to a robust product.

Likening a BCI device to something as complex as a smart phone or tablet makes clear that current BCI frameworks behave more like complete operating systems, requiring considerable expertise to develop. Following this analogy, the vast thousands of smart phone and tablet "apps" have proliferated precisely because manufacturers made simple, compact API available to develop software in isolation from the complexities of modern computational devices. Authors of apps now range from the highly experienced, who are capable of writing operating system-level programs (cf. current BCI developers), to the newly interested, who have a limited understanding of only the available programming interface.

The framework described here borrows the themes of computing standards and open development from prior efforts, including BCI2000, but is aimed at shifting the development focus away from academic research laboratories toward a larger, more casual developer base. It is hard to ignore the power of facilitating a crowd-sourcing paradigm in which developers with no previous BCI experience can contribute to a BCI development effort.

II. PROPOSED FRAMEWORK IMPLEMENTATION

The proposed framework is divided currently into two modules:

- **Core backend:** for handling data acquisition, system initialization and intermodule communication
- **Developer "app" API:** for providing a controlled environment in which developers of all backgrounds can contribute to BCI research and products.

A. Core backend

A key framework design feature is the separation of backend processing from the main app development space. Backend processes are implemented and obscured from downstream processing and provide functionality for coordinating the activities of data acquisition systems, intermodule communication and initialization of feedback streams (e.g. graphical displays, sound buffers or haptic devices). We have focused the current version implementation on a BCI protocol involving display of flickering visual stimuli for eliciting a steady state visual evoked response (SSVEP), visual feedback of BCI performance and interfacing with commercially available mobile EEG systems. We chose an initial SSVEP-based BCI application to test our development framework since the computational demands this type of BCI are representative of the complexities observed in other control protocols (e.g. motor imagery, auditory evoked potentials or slow cortical potentials). The proposed framework is independent from choice of elicited EEG potentials and related eliciting stimuli (if applicable, cf. motor imagery).

1) SSVEP—In electroencephalography (EEG), the SSVEP is an oscillating scalp potential observed over visual cortex regions that are evoked from a visual strobe stimulus. Specifically, when presented for a sufficient amount of time, the occipital electrode EEG response entrains to the strobe frequency and can be identified by the presence of the stimulus frequency and multiple harmonics in a spectral analysis. In BCI applications, an SSVEP stimulus is often associated with items for use in discrete choice selection. For example, a group of four visually presented items can be associated with four different SSVEP stimuli with different properties, often different strobe frequencies [2] or phase delays [3], [4]. Attending to one stimulus or another will elicit an SSVEP with response properties associated with the attended stimulus [5]; allowing for identification of the attended stimulus and item through classification methods. In this way many stimulus-item pairs can be associated and employed in a BCI design (e.g. 13–64 stimulus-item pairs) [6], [7].

2) Paradigm interface—In the proposed framework, all handling of graphical objects, including initialization and draw scheduling, are handled by a set of core routines that are unseen by application developers (see Fig. 1). Each SSVEP stimulus-item pair is associated with an identifier used in the brain-activity classifier for control of the BCI device. For instance, in an SSVEP paradigm with 4 stimulus-item pairs, each can be labeled with the ordinal numbers 1–4. The brain activity classifier, or decoder, works in parallel to the display and feedback mechanisms, but outputs the same ordinal number set for selection of application specific choices. In a motor-imagery system, the system can be designed to associate different classes of imagery with known stimulus-item identifiers (in this case stimuli are self-generated modulations to the sensorimotor rhythm) for brain-based control. A putative visual display in this case would allocate the entire screen for application visual feedback. A further modification, in the case of auditory feedback devices, would deallocate all visual displays, and instead manage the onboard sound card buffers for acoustic output.

3) Backend functionality—Three major underlying functions that require more sophisticated programming experience not intended for crowd-sourcing development are implemented in the proposed framework. These functions include:

- **System timing** – for synchronization of inputs, presentation of all stimuli for evoked brain response (if applicable, cf. motor-imagery) and scheduling feedback.
- **Inter-module communication** – for receiving and transmitting acquired signals, decoded responses and feedback signals.
- **App switching** – permits rapid switching to apps for performing different functions (e.g. communication, environmental control or web browsing) without closing and opening new programs.

We also include a simulation package for emulating the response of a decoding algorithm, highlighting the separation of neurological data acquisition from the app development process. The simulator replaces all of the modules on the left side of Figure 1.

4) EEG acquisition—Included in the proposed framework are a description and sample implementations of standard interface methods for EEG acquisition systems. In general, these methods include: 1) open a port to the device, 2) connect, 3) start acquisition and 4) read data. We have pre-implemented Python wrapping classes based on this functionality for the following mobile acquisition systems:

1. g.tec g.MOBILab+
2. Emotiv Epoc
3. Neurosky Mindband
4. Custom serial port/bluetooth communication (e.g. Arduino communication)

The Emotiv Epoc, Neurosky Mindband and Arduino-coupled electrophysiology acquisition systems are relatively affordable for individual hobbyists and start-up device manufacturers alike. This framework, with pre-implemented hardware interfaces and standard hardware

API, will facilitate development of BCI applications by reducing the knowledge overhead needed to interface with these complex devices in real-time.

B. Developer "app" API

The developer app API is the primary coding entrance point for the intended audience of casual BCI app developers. We have devised a simplified and concise programming standard consisting of just two required methods:

1. draw()
2. update(choice, select)

Both methods are scheduled by the backend paradigm controller (see Fig. 1) and called once per update cycle. They are handled in a separate process from data acquisition and brain response classification in which the update cycle runs as fast as possible, but is often constrained by the display frame refresh rate (e.g. 60 Hz or 75 Hz for most LCD monitors). Details for these two requisite methods are given as Python-code descriptions in Figure 3.

III. PROTOTYPE TRIAL

We have completed the initial stage of prototype testing with a subject (45 yo, male) having amyotrophic lateral sclerosis (ALS). All methods were approved by the Boston University Institutional Review Board and informed consent was obtained by the participant and his legal guardian prior to the experimental session. Three four-choice SSVEP applications were developed using the proposed framework and executed on a Lenovo x220 laptop with 4 GB onboard memory and an Intel 64-bit Core i5 processor running Ubuntu 11.10 Linux. The goal of this trial was to: 1) ensure the system was capable of running in real time with negligible system lag, and 2) verify proper stimulus presentation, signal acquisition and signal analysis for use in BCI applications.

A. Overall system performance

The system is designed to run with processing times (or system delays) below the effective sampling rate of stimulus presentation and feedback. Figure 4 shows the overall system delay for a segment of application running time as a time-series representation of system delay labeled according to application "mode." In this example, the application started in an initial app menu state then switched into our speech-output application which has multiple levels of functioning (initial level 1, middle section of Fig. 4; and a more specific level 2, right section of Fig. 4. These data illustrate the application incurs < 5 ms average system delay which is less than the overall processing interval, though certain functions require more processing than others.

B. User trial

Here we report on the verification of proper stimulus presentation combined with signals acquisition through a proxy of spectral analysis. This analysis is based on a pretrial calibration period, in which the subject was instructed to attend to one of four SSVEP stimuli placed around the border of the laptop screen (see Fig. 2, bottom left). Each stimulus was flashing at a different rate (12 Hz, 13 Hz, 14 Hz and 15 Hz, respectively). Figure 5 shows the average power spectral density (PSD) across all trials for each stimulus attention condition. According to the properties of SSVEP, we expect increased PSD amplitude for the specified strobe frequencies during appropriate task conditions. This result is confirmed in Figure 5 where local peaks in PSD amplitude are observed for the 12 Hz (blue), 13 Hz (green), 14 Hz (red) and 15 Hz (purple) conditions. From these results, we can infer the system is operating within acceptable tolerances and is capable of eliciting and recording valid neurological data.

IV. NEXT STEPS

A. Iterative adjustments based on user feedback

Our experience evaluating this framework as a practical BCI system for users with movement and speech impairments has lead to new design considerations and constraints. Two major requested features are:

1. Real-time feedback of BCI activity *between* selections (e.g. more salient visual representations of current decision)
2. More flexible selection methods (e.g. recorded available EMG activity, single vs. multiple eye blinks)

Refinements such as these highlight the importance of including potential users with impairments in the design process, and a flexible software architecture to accommodate changing device specifications.

B. Tablet implementation

A pure Python implementation was chosen for the proposed framework given recent developments in native Python scripting on Android devices. A number of Python environments now exist for use in the Android OS including both standard Python computing and graphical user interface development. Our framework has been designed from the start with these in mind, with an eye toward near-future implementations on commercial mobile computing platforms.

C. Context-aware adaptation

We propose a robust, mobile platform capable of integrating multiple environmental signals into a context-aware communication system. Examples of this type of computational approach have been explored previously for non-communication BCI systems (e.g. wheel chair navigation [8]) and non-BCI augmentative and alternative communication (e.g. iconCHAT [9]), but have not yet been integrated for a BCI communication device. Further, these two examples include just one additional context feature each, as does the more traditional context-aware methods of word and language prediction. To expand the environmental feature set, we include:

- **GPS f**or access to location-specific commands
- **Calendar** for event- and time-specific inputs
- **Audio** for Google speech-to-text input
- **Bluetooth** for receiving EEG/EOG BCI-specific selection commands
- **Video** for object/face recognition from built-in cameras

We plan to use these features in future implementations of the proposed framework, providing an adaptive user interface capable of presenting subjects with the most likely conversation options in a given contextual situation. This system should be definable at multiple levels of conversation including phrase selection and word/letter spelling with conversation-, language- and word-level prediction. The goal is to reduce the number of selections necessary for user conversation and improve on effective information transfer rate through computational assistance.

Acknowledgments

This work was supported in part by CELEST, a National Science Foundation Science of Learning Center (NSF OMA-0835976) and the National Institute on Deafness and Other Communication Disorders (NIDCD R03-DC011304)

References

1. Schalk G, McFarland DJ, Hinterberger T, Birbaumer N, Wolpaw JR. BCI2000: a general-purpose brain-computer interface (BCI) system. IEEE Transactions on Biomedical Engineering. 2004; 51(6):1034–1043. [PubMed: 15188875]
2. Parini S, Maggi L, Turconi AC, Andreoni G. A robust and self-paced BCI system based on a four class SSVEP paradigm: algorithms and protocols for a high-transfer-rate direct brain communication. Computational Intelligence and Neuroscience. 2009:864564. [PubMed: 19421416]
3. Gysels E, Celka P. Phase synchronization for the recognition of mental tasks in a brain-computer interface. IEEE Transactions on Neural Systems and Rehabilitation Engineering. 2004; 12(4):406–415. [PubMed: 15614996]
4. Bin G, Gao X, Wang Y, Li Y, Hong B, Gao S. A high-speed BCI based on code modulation VEP. Journal of Neural Engineering. 2011; 8(2):025015. [PubMed: 21436527]
5. Morgan ST, Hansen JC, Hillyard SA. Selective attention to stimulus location modulates the steady-state visual evoked potential. Proceedings of the National Academy of Sciences of the United States of America. 1996; 93(10):4770–4774. [PubMed: 8643478]
6. Cheng M, Gao X, Gao S, Xu D. Design and implementation of a brain-computer interface with high transfer rates. IEEE Transactions on Biomedical Engineering. Oct; 2002 49(10):1181–1186. [PubMed: 12374343]
7. Sutter EE. The brain response interface: communication through visually-induced electrical brain responses. Journal of Microcomputer Applications. Jan; 1992 15(1):31–45.
8. Vanacker G, del J, Millán R, Lew E, Ferrez PW, Moles FG, Philips J, Van Brussel H, Nuttin M. Context-based filtering for assisted brain-actuated wheelchair driving. Computational intelligence and neuroscience. Jan.2007 2007:25130. [PubMed: 18354739]
9. Patel R, Radhakrishnan R. Enhancing access to situational vocabulary by leveraging geographic context. Assistive Technology Outcomes and Benefits. 2007; 4(1):99–114.

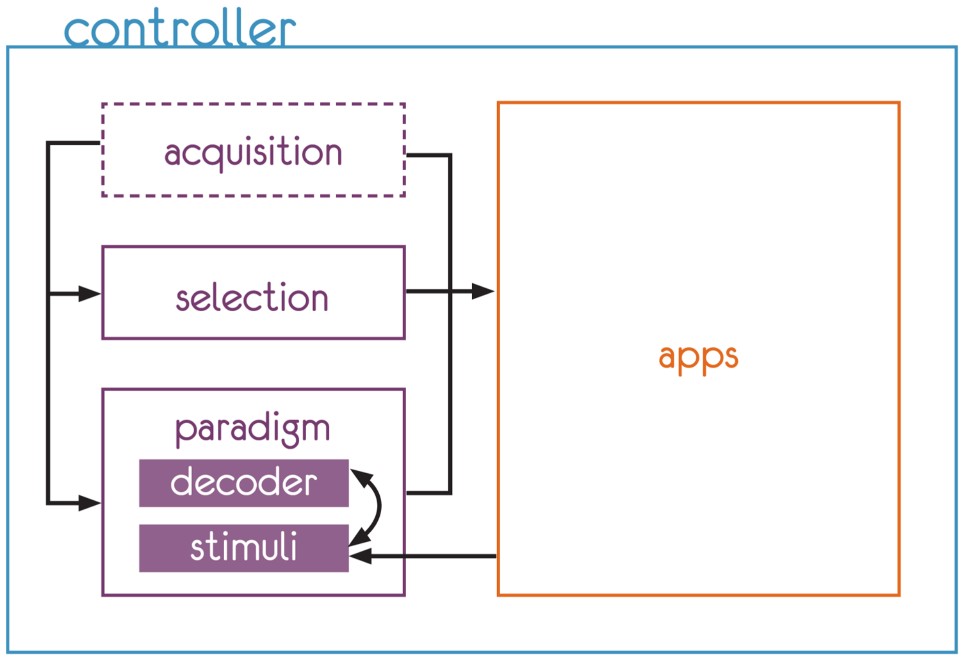

Fig. 1.
A system diagram of the framework processing modules. Arrows indicate lines of communication between processing modules, handled in the background by a master controller. The effort here has been to isolate, as much as possible, the app environment from the remaining modules. This isolation is key for invoking a “crowd-sourcing” paradigm for BCI application development.

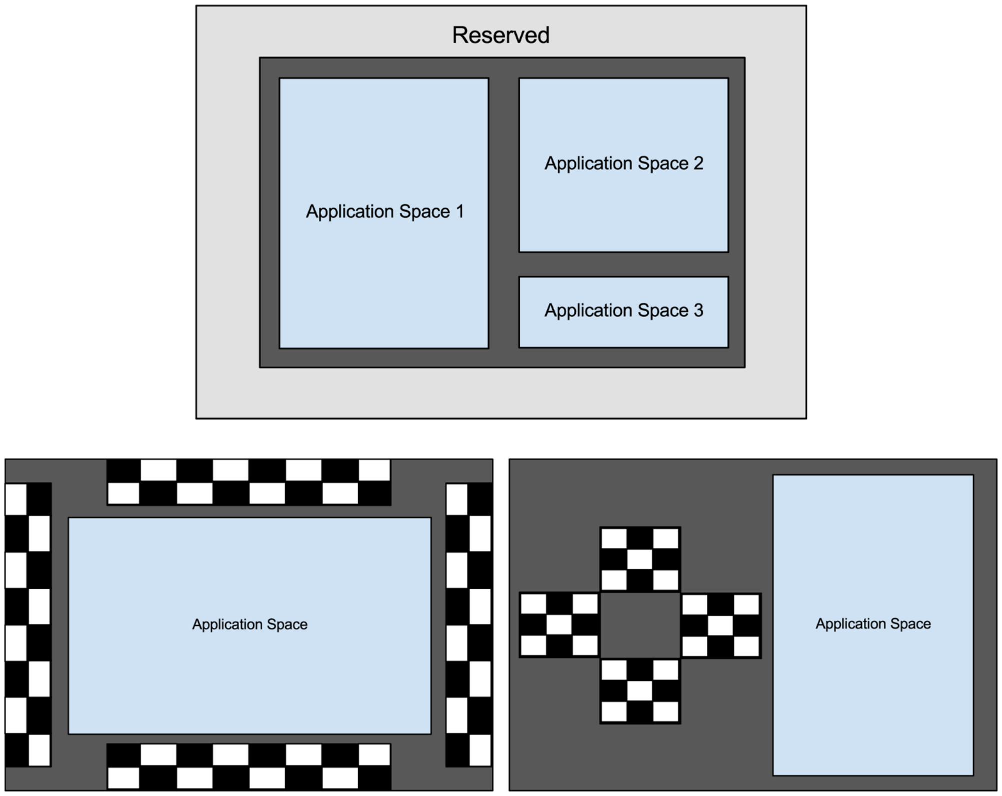

Fig. 2.
Examples of graphical layouts that can be implemented using the proposed framework. (Top) A general description of visual layouts. A section of the screen (e.g. the border in this example) is reserved for stimuli and the remaining area is partitioned for multiple apps, or displays. (Bottom) Example layouts employed in existing framework applications: (left) Screen-border SSVEP layout with four different stimuli and a single application space; (right) Split-screen SSVEP layout, again with four stimuli and a half-screen application space.

```
def draw():
# Place all drawing or multimodal output
# commands here

def update(choice = None, select = False):
# Updates all methods based on current
# decoding choice
#
# \textbf{choice}: a scalar value part of an
#         enumerated type known to both
#    decoders and feedback methods
#
# \textbf{select}: a boolean value indicating
#         whether to "select" or
#    act upon the [choice] value
```

Fig. 3.
Code description for draw() and update() methods required for all app implementations. The update function assumes no choices have been made, and, therefore, should not process any selections.

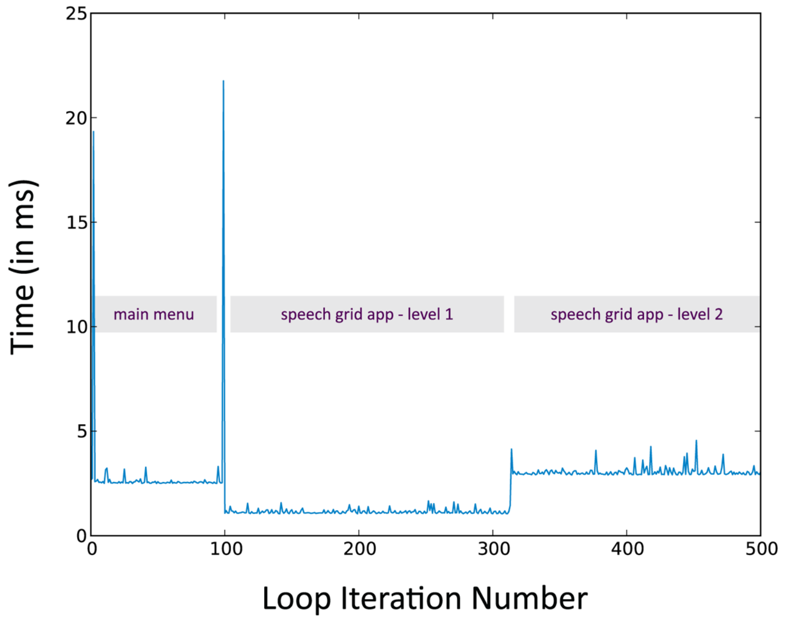

Fig. 4.
Overall system lag for an example implementation employing the proposed framework displayed as system delay evolution over time. In this example, the BCI application entered a toplevel main menu, then entered a hierarchical speech output system with two layers. Each app (in. main menu, or speech output) utilized different levels of resources, though overall system delay averaged less than 5 ms.

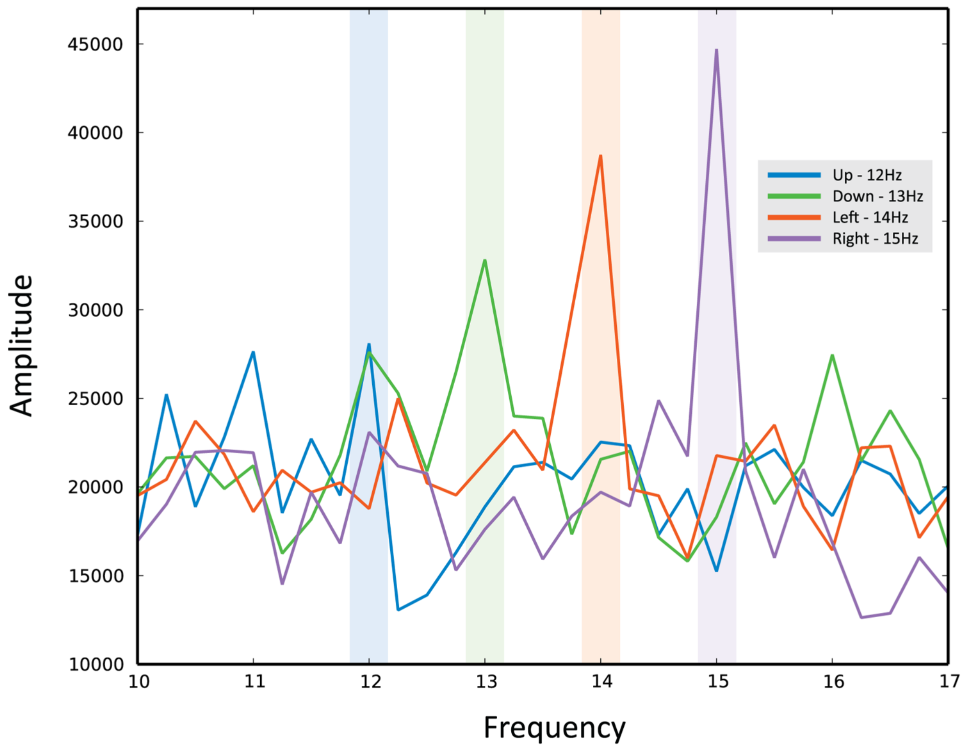

Fig. 5.
Sample power spectrum during 4-choice SSVEP task.